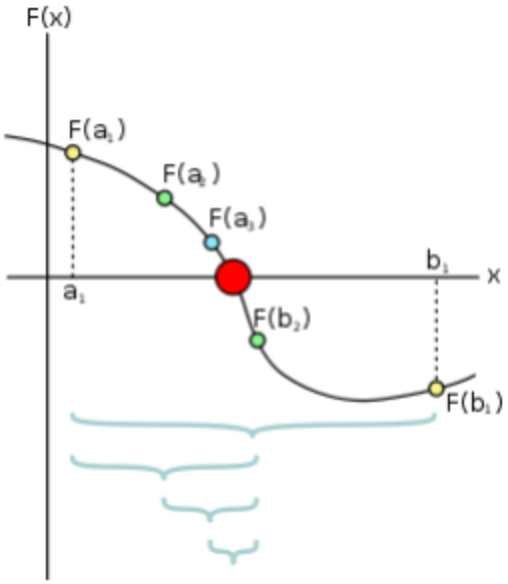


B. Bisection Method

C

Uğurkan Ateş 151044012 .

Embedded Systems 433 Project Report.

This is bisection method. Its mostly used for finding root of one unknown equations.

"**Bisection Method** = a numerical method in Mathematics to find a root of a given ... The **Bisection Method** is given an initial interval [a..b] that contains a root

"

I implemented on Logisim, created C code, created State table, state diagram and put on expressions according to inputs(boolean simplify)

It took 1.5 week to develop this project (along with 0.5 week for GCD HW0)

It took me 20(ish) try on to get fully right. Reasons for this at %10 of my datapath (states) being not fully viable for every step of C Code, %30 of hardness of putting state diagram / circuit / boolean's to Logisim control module I created , %60 of LOGISIM program not working correctly.(getting RED wire on places that shouldnt be RED, auto wiring each input of D-Flip flops so creating major errors that kills your all hope.)

I also struggled with IEEE754 jar library at first . I was developing inputs as pin so after discovering it had constant inputs my whole design had to change(because of pin 32 bit multiply situtation) I was using different bits for inputs and it was limited. Also I didn't know it had probe so I was struggling for 2 days straight.

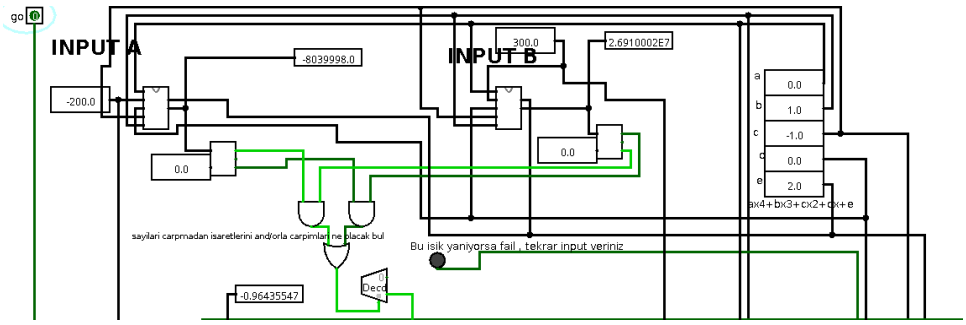
My modules in Project

1-Main = main module takes inputs of A,B,C,D,E and X of Ax^4 +Bx^3+Cx^2+Dx'1 + Ex'0 is my equation

Then it calculates if they are okay inputs .

Just like this.

```
 If (func(a1) * func(b1) >= 0)  //bizim icin bas true olmasi kucuktur olmasi yani
  {
     printf("You have not assumed right a and b\n");
     return;
  }
```



I did a smart design choice here. Instead of multiplying like C code I instead looked into Signatures of function calls. Therefore I am removing possibility of overflow and unnecessary calculation. Very smart design. I used this in multiple parts of project. According to first part check I put in to decoder. At first most 20ish try 0 input was actually connected to LED to indicate its not correct inputs but I then changed to only state 2 to connect LED. (because led is starting light if we don't do that)

```
   double c = a1;  // floating
```

A = a_input

B= b _ input

I do these to get them on registers. C 's initial value is just A value. I assign A from input to C at first (because A register doesn't have A value in that time)

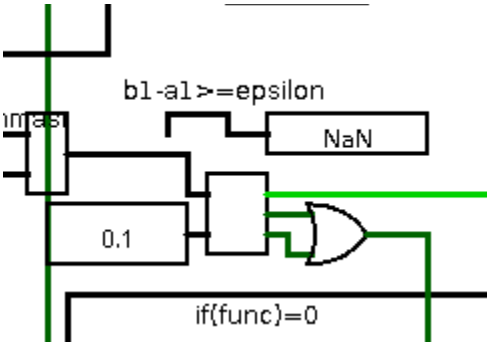
Then it comes to while checking part

while ((b1-a1) >= EPSILON)

{

..

I have to compare them with EPSILON . EPSILON is inputtable value here.



In this FP compare module middle like normal comparison means equal. But since this is signed last one means top > bottom input , first one means top < bottom input . This is %100 negative of normal comperator inside logisim . Again this was also baffling I had to check actual library code.

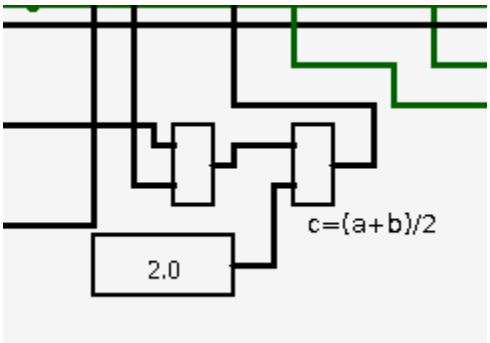
Very bad decision from them indeed. Not even putting basic labels to input / output. I actually said them and probably gonna update a code fix for them.

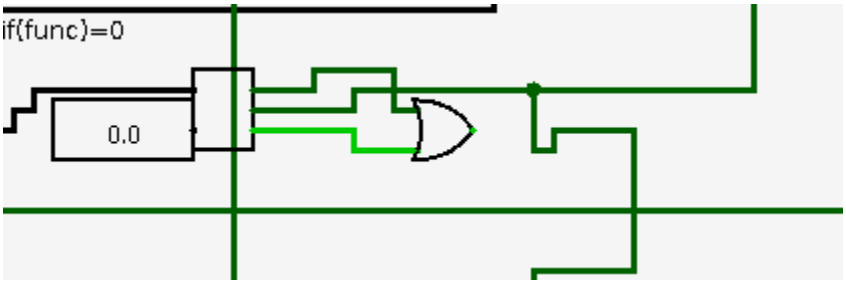
Then it comes to part

// Find middle point

c = (a1+b1)/2;

Its basic 2 arithmetic unit module equation. I used adder and dvision .





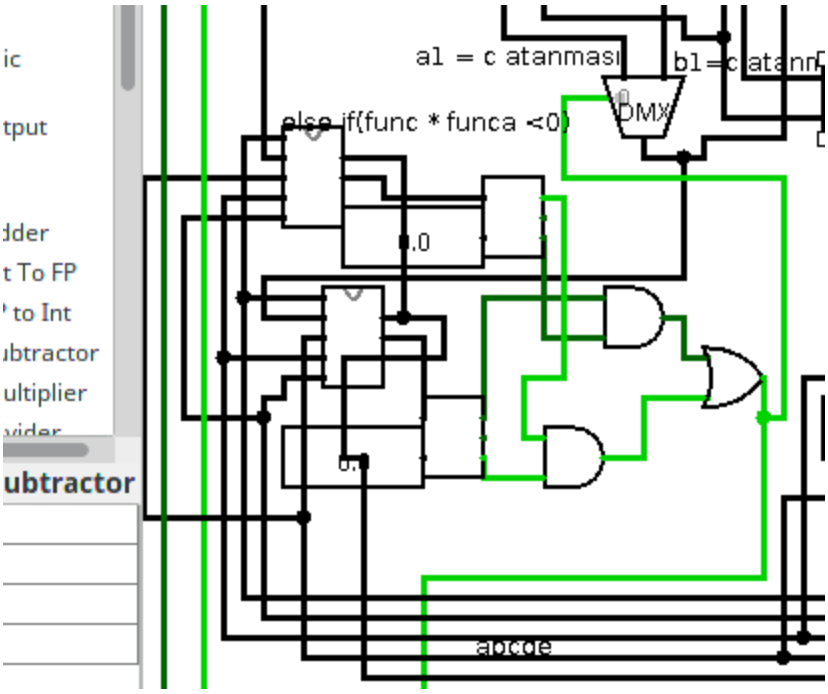
// Check if middle point is root

```
if (func(c) == 0.0)
    break;
```

I did this as separate because if true it goes STATE 9 (print)

```
// Decide the side to repeat the steps
if (func(c)*func(a1) < 0)
    b1 = c;
else
    a1 = c;
```

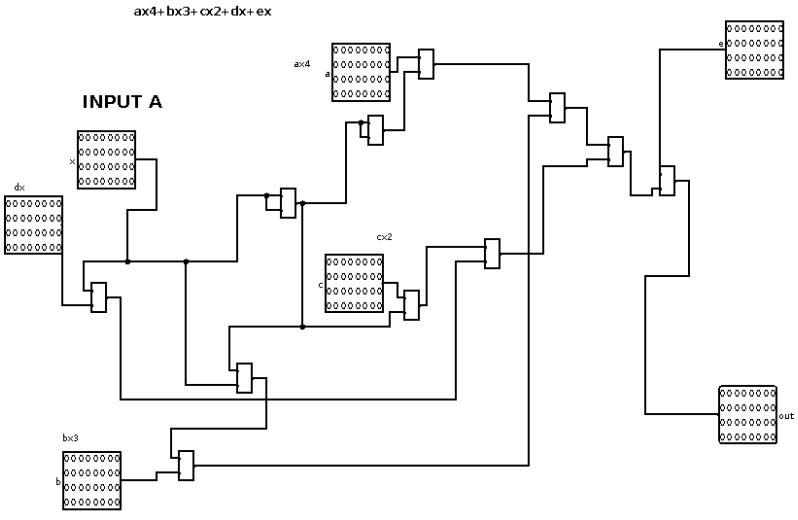
Then these



. This is most general outline of main module

2- Controllable function module

This is implementation of Ax4+bx3+dx2+cx+e module.



Nothing too hard / worth mentioning. Just they are 32 bits in module but inputs are FP constants so I don’t convert them with FP INT -> FLOAT thing

3- Control Unit

Main implementation of STATE table. I used 10 state

I used 5 input signals

1- Go signal = if 0 circuit wont exit 0000 state it will come back. If 1 continue. Have to close to see results if CLOCK is too fast.

2-Bas signal = if inputs are not wrong(results of func calls multiplication should be < 0 if > = 0 then signal is set 0 , else its 1 .

3- BEPS = ıt controls while loops condition B-A >= epsilon . Looks register values and subs then comparator starts with epsilon value. IF True loop continues(>= eps means true) else it goes STATE 9 Print

4- func signal = if(func c = =0) it means it found root and go print. Else continue . !FUNC needs for continue on loop

5- AC signal = ac signal is true if

else if (func(c)*func(a1) < 0)

Is true . else is

0000 = State 0 waiting state goes accordingly GO signal

0001 = İf Go signal is 1 then its here. Next state is according to BAS either 0010(state2)

Or state 3 for normal continuation(0011)

0010 = print say its wrong say take some inputs and goes 0000 in next clock

0011 = do assigments of Register a=ainput b = biput c = ainput . Then go 0100 (state4)

0100 = go state 9 print if epsilon is bigger than b1-a1 registers. else just go 0101

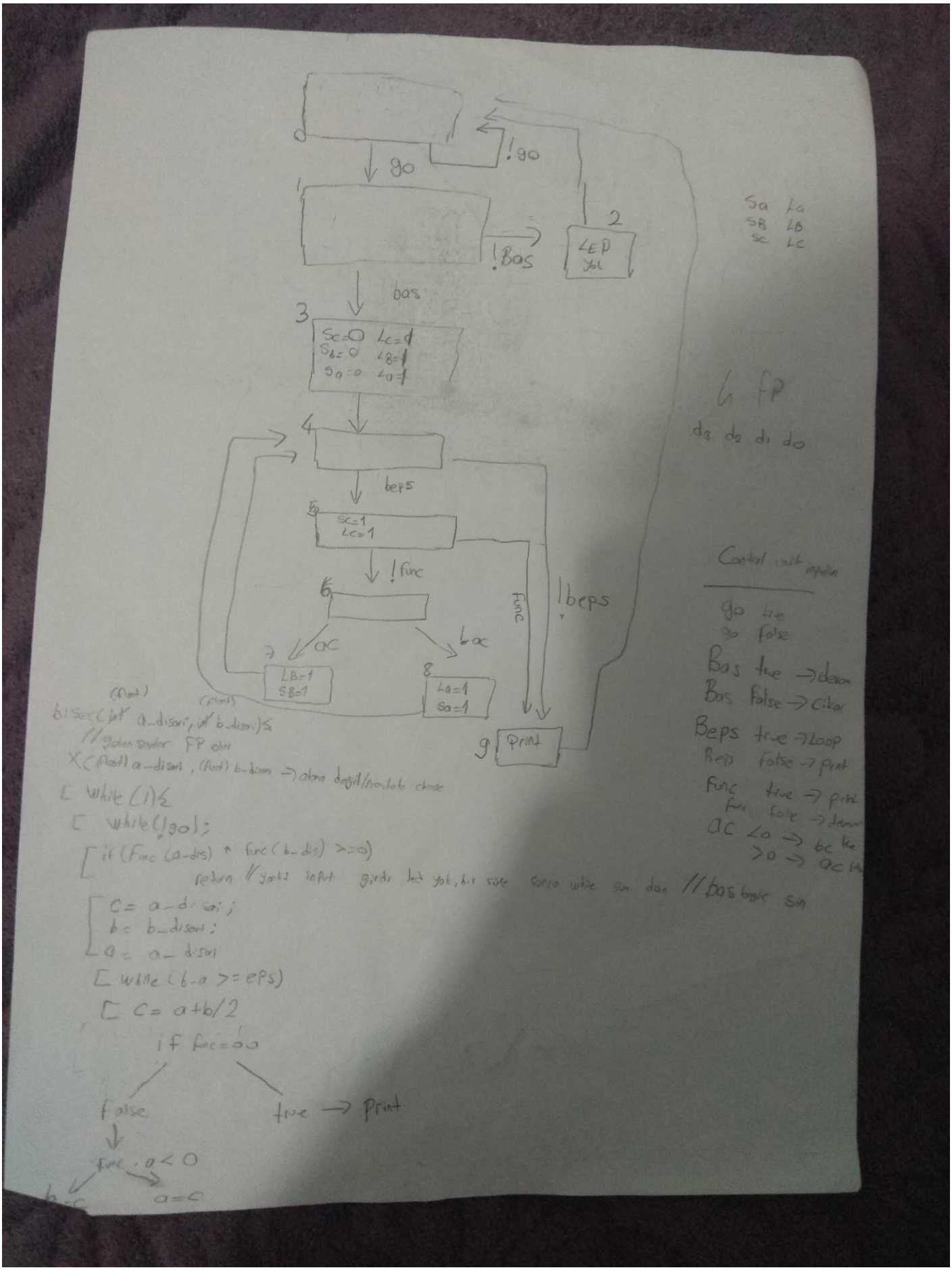
0101 = do c= (a+b) / 2 then go next state according to FUNC sgnal if func is true go state9

0110= go 7 or 8 according to else if . (AC signal)

0111= do b= c registers and go state 4.

1000 = do a=c and go state 4

1001 Print and go state 0


go
!go
!Bas
2
LEP
Yok
bas
3
Sc=0 Lc=1
SB=0 LB=1
Sa=0 La=1
4
beps
5
Sc=1
Lc=1
!func
6
func
!beps
ac
bac
7
LB=1
SB=1
8
La=1
Sa=1
9
Print
Sa La
SB LB
Sc Lc
FP
d3 d2 d1 d0
Control unit inputları
go true
go false
Bas true → devam
Bas false → cikar
Beps true → Loop
Beps false → print
Func true → print
Func false → devam
ac La → bc ke
>0 → ac ke
(float) (float)
bisec(int a_disari, int b_disari)
//global scope FP olur
X((float) a_disari, (float) b_disari) → aslında degil/translate olunce
{ while(1){
{ while(!go);
{ if(Func(a_dis) * func(b_dis) >=0)
return //yanlis input girdi bet yok, bir süre sonra while yine dön //bas basic sin
{ c= a_disari;
b= b_disari;
a= a_disari
{ while(b-a >= eps)
{ c= a+b/2
if func=0
false
true → print
func. a<0
b=c
a=c

4 FF-flip

P_3 a	P_2 b	P_1 c	P_0 d	Inputs G_0 e	P_{GS} f	B_{eps} g	F_{ac} h	A_c k	N_3	N_2	N_1	N_0
0	0	0	0	1	-	-	-	-	0	0	0	1
0	0	0	0	0	-	-	-	-	0	0	0	0
0	0	0	1	-	0	-	-	-	0	0	1	0
0	0	0	1	-	1	-	-	-	0	0	1	1
0	0	1	0	-	-	-	-	-	0	0	0	0
0	0	1	1	-	-	-	-	-	0	1	0	0
0	1	0	0	-	-	1	-	-	0	1	0	1
0	1	0	0	-	-	0	-	-	1	0	0	1
0	1	0	1	-	-	-	0	-	0	1	1	0
0	1	0	1	-	-	-	1	-	1	0	0	1
0	1	1	0	-	-	-	-	1	0	1	1	1
0	1	1	1	-	-	-	-	0	1	0	0	0
1	0	0	0	-	-	-	-	-	0	1	0	0
1	0	0	1	-	-	-	-	-	0	1	0	0
				-	-	-	-	-	0	0	0	0

```
#define EPSILON 0.00001  // 0,1 og kod.
#include <stdio.h>

double func(double x)
{
   return x*x*x - x*x + 2;
}

void bisection(int a, int b)
{
   double   a1 = a; //sayıları extend edip FP yap
   double b1 = b; //bu alttakini hangi sirada cagiririz bilmiyom ilk cagirabiliriz intle
   if (func(a1) * func(b1) >= 0)  //bizim icin bas true olmasi kucuktur olmasi yani
   {
      printf("You have not assumed right a and b\n");
      return;
   }

   double c = a1;  // floating
   while ((b1-a1) >= EPSILON)
   {
      // Find middle point
      c = (a1+b1)/2;

      // Check if middle point is root
```

```
        if (func(c) == 0.0)
            break;

        // Decide the side to repeat the steps
        else if (func(c)*func(a1) < 0)
            b1 = c;
        else
            a1 = c;
    }
    printf("The value of root is : %f\n",c);
}

int main()
{
    // Initial values assumed
    int a =-200, b = 300;
    bisection(a, b);
    return 0;
}
```

P3 flip flopu

```
(!a * b * !c * !d * !g) +
(!a * b * !c * d * h) +
(!a * b * c * !d * k)
```

P2 flip flopu

```
(!a * b * !c * !d *g) +
(!a * b * !c *d * !h) +
```

(!a * b * c !d * k) +

(!a * c * d) +

(a * !b * !c *!d)

P1 flip flopu

(!a * !b * !c *d) +

(!a *b * !c * d * !h)+

(!a *b *c *!d * k) +

P0 flip flopu

(*!a *!b *!c *!d * e)+

(*!a *!b *!c *d *f) +

(*!a *b *!c *!d) +

(!a *b * !c * d *h) +

(!a * b * c !d *k)